```
import numpy as np

def skws(v):
    if len(v) != 3:
        raise ValueError('O vetor de entrada deve ter 3 elementos.')

    S = np.array([
        [0, -v[2], v[1]],
        [v[2], 0, -v[0]],
        [-v[1], v[0], 0]
    ])
    return S



def Rw(phi, theta):
    return np.array([
        [1, 0, np.sin(theta)],
        [0, np.cos(phi), np.sin(phi)*np.cos(theta)],
        [0, np.sin(phi),  np.cos(phi)*np.cos(theta)]
    ])

def dRw(phi, theta, dphi, dtheta):
    return np.array([
        [1, 0, np.cos(theta)*dtheta],
        [0, -np.sin(phi)*dphi, dphi*np.cos(phi)*np.cos(theta)-np.sin(phi)*np.sin(theta)*dtheta],
        [0, np.cos(phi)*dphi,  -dphi*np.sin(phi)*np.cos(theta)-np.cos(phi)*np.sin(theta)*dtheta]
    ])




import numpy as np

def calcular_coordenadas(gama, r, altura=0.0):
    """
    Calcula as coordenadas dos pontos para uma estrutura circular.

    Parâmetros:
    gama (float): Ângulo inicial em graus.
    r (float): Raio da estrutura em metros.
    altura (float): Altura dos pontos em metros (padrão é 0.0).

    Retorna:
    numpy.ndarray: Array 3D com as coordenadas dos pontos.
    """
    # Cálculo dos ângulos para a estrutura
```

```
    pis = [gama, 120 - gama, 120 + gama, -120 - gama, -120 + gama, -gama]

    # Convertendo graus para radianos
    pis_rad = np.radians(pis)

    # Calculando as coordenadas x e y para os pontos
    x = r * np.cos(pis_rad)
    y = r * np.sin(pis_rad)
    z = np.full(len(x), altura)  # array de altura constante

    # Agrupando as coordenadas em um array 3D
    coordenadas = np.array([x, y, z])

    # Transposição para que cada coluna represente um ponto
    return coordenadas

# Exemplo de uso da função para a base
gama_B = 15  # Ângulo inicial em graus
r_B = 0.6  # Raio da base em metros
B = calcular_coordenadas(gama_B, r_B)

print("Coordenadas Bi:")
print(B)

# Exemplo de uso da função para a plataforma
gama_P = 45  # Ângulo inicial em graus
r_P = 0.25  # Raio da plataforma em metros
altura_P = 0.0  # Altura da plataforma em metros
P = calcular_coordenadas(gama_P, r_P, altura_P)

print("\n\n")
print("Coordenadas Pi:")
print(P)




import numpy as np

def rotX(theta):
    return np.array([
        [1, 0, 0],
        [0, np.cos(theta), -np.sin(theta)],
        [0, np.sin(theta), np.cos(theta)]
    ])
```

```
def rotY(theta):
    return np.array([
        [np.cos(theta), 0, np.sin(theta)],
        [0, 1, 0],
        [-np.sin(theta), 0, np.cos(theta)]
    ])

def rotZ(theta):
    return np.array([
        [np.cos(theta), -np.sin(theta), 0],
        [np.sin(theta), np.cos(theta), 0],
        [0, 0, 1]
    ])

def cinemática_inversa(trans, rotation, home_pos, R, P, B):
    """
    Calcula as variáveis de cinemática inversa para uma plataforma.

    Parâmetros:
    trans (numpy.ndarray): Vetor de translação.
    rotation (numpy.ndarray): Vetor de rotação.
    home_pos (numpy.ndarray): Posição inicial da plataforma.
    R (numpy.ndarray): Matriz de rotação.
    P (numpy.ndarray): Matriz de coordenadas dos pontos da plataforma.
    B (numpy.ndarray): Matriz de coordenadas dos pontos da base.

    Retorna:
    tuple: l, lll, L, s
    """
    # Allocate for variables
    l = np.zeros((3,6))
    lll = np.zeros((6))

    # Calcular os comprimentos das pernas l
    l = np.repeat(trans[:, np.newaxis], 6, axis=1) + np.repeat(home_pos[:, np.newaxis], 6,
axis=1) + np.matmul(R, P) - B

    # Calcular os comprimentos lll
    lll = np.linalg.norm(l, axis=0)

    # Calcular as posições das pernas no referencial global L
    L = l + B

    # Calcular os vetores unitários s
    s = l / lll

    return l, lll, L, s
```

```
# Exemplo de uso da função
# Definições dos parâmetros
trans = np.array([0, 0, 1])
rotation = np.array([0, 0, 0])
home_pos = np.array([0, 0, 1])

# Calculando as coordenadas para a base e a plataforma
gama_B = 15  # Ângulo inicial em graus
r_B = 0.6  # Raio da base em metros
B = calcular_coordenadas(gama_B, r_B)

gama_P = 45  # Ângulo inicial em graus
r_P = 0.25  # Raio da plataforma em metros
altura_P = 0.0  # Altura da plataforma em metros
P = calcular_coordenadas(gama_P, r_P, altura_P)

# Calcular a matriz de rotação total
R = np.matmul(np.matmul(rotX(rotation[0]), rotY(rotation[1])), rotZ(rotation[2]))

# Calculando os valores de cinemática inversa
l, lll, L, s = cinemática_inversa(trans, rotation, home_pos, R, P, B)

#print( l)
#print("\n")
#print( lll)
#print("\n")

#print('Posições das pernas no referencial global (L):', L)
#print("\n")

print( s)



e1 = 0.5
e2 = 0.5
m1 = np.array([0.1] * 6)
m2 = np.array([0.1] * 6)
I1ii = np.array([[[0.00625, 0, 0], [0, 0.00625, 0], [0, 0, 0]]] * 6).transpose((1, 2, 0))
I2ii = np.array([[[0.00625, 0, 0], [0, 0.00625, 0], [0, 0, 0]]] * 6).transpose((1, 2, 0))
Ixx=0.00625*2



mp = 30
Ipb = np.array([
    [0.7813, 0, 0],
```

```
    [0, 0.7813, 0],
    [0, 0, 1.562]
])

fe = np.zeros(3)
ne = np.zeros(3)


Mi = np.zeros((6,3,3))
Ci = np.zeros((6,3,3))
Gi = np.zeros((6,3))
Ixx = 0.00625*2    # Aqui na verdade é um Ixxi na referencia da perna



mce = np.zeros(6)
mge = np.zeros(6)
mco = np.zeros(6)
g = np.array([0,0,-9.807])
for i in range(6):
    mce[i] = (1 / (lll[i]**2)) * (m1[i] * (e1**2) + m2[i] * (lll[i]-e2)**2)
    mge[i] = (1 / (lll[i])) * (m1[i] * (e1) + m2[i] * (lll[i]-e2))
    mco[i] = (1 / lll[i]) * m2[i] *(lll[i] -e2) - (mce[i]+Ixx/(lll[i])**2)


times = np.linspace(0, 10, 500)

# Amplitudes dos atuadores
amplitudes = np.zeros((6, len(times)))
# Calculando amplitudes ao longo do tempo
for k, t in enumerate(times):
    zz = 0.3 * np.sin(3 * t)
    dzz_dt = 0.3 * 3 * np.cos(3 * t)
    ddzz_dt = -0.3 * 3**2 * np.sin(3 * t)


    trans = np.transpose(np.array([0, 0, zz]))  # X, Y, Z
    dtrans = np.transpose(np.array([0, 0, dzz_dt]))  # X, Y, Z
    ddtrans = np.transpose(np.array([0, 0, ddzz_dt]))  # X, Y, Z




    phi =np.zeros_like(t)
    dphi =np.zeros_like(t)
```

```
    ddphi =np.zeros_like(t)

    theta =np.zeros_like(t)
    dtheta =np.zeros_like(t)
    ddtheta =np.zeros_like(t)

    rotation = np.transpose(np.array([phi, theta, 0]))  # X, Y, Z
    drotation = np.transpose(np.array([dphi, dtheta, 0]))  # X, Y, Z
    ddrotation = np.transpose(np.array([ddphi, ddtheta, 0]))  # X, Y, Z

    wp = np.matmul(Rw(phi, theta), drotation)
    alphap = np.matmul(dRw(phi, theta, dphi, dtheta), drotation) + np.matmul(Rw(phi,
theta), ddrotation)


    x = np.concatenate((trans, rotation))
    dx = np.concatenate((dtrans, wp))
    ddx = np.concatenate((ddtrans, alphap))

    R = np.matmul(np.matmul(rotX(rotation[0]), rotY(rotation[1])), rotZ(rotation[2]))
    # Calculando os valores de cinemática inversa
    l, lll, L, s = cinemática_inversa(trans, rotation, home_pos, R, P, B)
    RP = np.matmul(R, P)
    Ji = np.zeros((6,3,6))
    dJi = np.zeros((6,3,6))
    Jp = np.zeros((6,6))

    Mi = np.zeros((6,3,3))
    Ci = np.zeros((6,3,3))
    Gi = np.zeros((6,3))

    mce = np.zeros(6)
    mge = np.zeros(6)
    mco = np.zeros(6)


    for i in range(6):
        mce[i] = (1 / (lll[i]**2)) * (m1[i] * (e1**2) + m2[i] * (lll[i]-e2)**2)
        mge[i] = (1 / (lll[i])) * (m1[i] * (e1) + m2[i] * (lll[i]-e2))
        mco[i] = (1 / lll[i]) * m2[i] *(lll[i] -e2) - (mce[i]+Ixx/(lll[i])**2)



    for i in range(6):
      # Vetor exemplo
        s_hat = s[:, i]
```

```
        # Produto externo de s_hat consigo mesmo
        ssT = np.outer(s_hat, s_hat)

        sx = skws(s_hat)
        sx2 = np.matmul(sx, sx)
        Ji[i] = np.hstack((np.eye(3), -skws(RP[:, i])))
        Jp[i] = np.matmul(s_hat.T, Ji[i])
        dxi = np.matmul(Ji[i], dx)
        #wi = 1/lll[i]*np.matmul(sx, dxi)
        dl = s_hat.T @ dxi
        Mi[i] = m2[i]*ssT -  (mce[i] + Ixx/lll[i]**2)*sx2
        Gi[i] = -np.matmul(( m2[i]*ssT -mge[i]*sx2), g)
        # Calculate the first operand for matrix multiplication as an array
        operand1 = (1/lll[i])*(2*mco[i]*dl +e2/lll[i]*m2[i]*np.matmul(s_hat,dxi.T))
        Ci[i] = -operand1 * sx2 # Multiply each element of sx2 by the scalar operand1


    Ip = R @ Ipb @ np.transpose(R)

    sumM = np.zeros((6, 6))
    sumC = np.zeros((6, 6))
    sumG = np.zeros(6)


    for i in range(6):
        dJi[i] = np.hstack((np.zeros((3, 3)), -skws(wp) @ skws(RP[:, i]) + skws(RP[:, i]) @
skws(wp)))

        sumM += Ji[i].T @ Mi[i] @ Ji[i]
        sumC += Ji[i].T @ Mi[i] @ dJi[i] + Ji[i].T @ Ci[i] @ Ji[i]
        sumG += Ji[i].T @ Gi[i]



    Mp = np.block([
          [mp * np.eye(3), np.zeros((3, 3))],
          [np.zeros((3, 3)), Ip]
      ])
    Cp = np.block([
          [np.zeros((3, 3)), np.zeros((3, 3))],
          [np.zeros((3, 3)), skws(wp) @ Ip]
      ])
    Gp = np.hstack((-mp * np.array([0, 0, -9.807]), np.zeros(3)))

    MX = Mp + sumM
    CX = Cp + sumC
    GX = Gp + sumG
```

```
    Fext = MX @ ddx + CX @ dx + GX

    # Calcula a inversa da matriz Jacobiana
    jac_inv = np.linalg.inv(Jp)

    # Calcula a transposta da inversa da matriz Jacobiana
    jac_inv_T = jac_inv.T

    tau = np.matmul(jac_inv_T, Fext)
    for j in range(6):
      #amplitudes[j, k] = lll[j]
      amplitudes[j, k] = tau[j]




import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation

# Config para a animação
fig = plt.figure(figsize=(12, 10))

#Grafico 3D
ax3d = fig.add_subplot(311, projection='3d')

ax3d.set_xlim3d(-1, 1)
ax3d.set_ylim3d(-1, 1)
ax3d.set_zlim3d(0, 1.4)

ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')

#Grafico 2D - Deslocamento
ax2d = fig.add_subplot(323)

ax2d.set_xlim(0, 10)
ax2d.set_ylim(0.5, 1.5)

ax2d.set_xlabel('Time (s)')
ax2d.set_ylabel('Deslocamento (m)')

x_line, = ax2d.plot([], [], lw=2, label='x', color='blue')  # Linha x em azul
```

```
y_line, = ax2d.plot([], [], lw=2, label='y', color='green')  # Linha y em verde
z_line, = ax2d.plot([], [], lw=2, label='z',
                    color='red')  # Linha z em vermelho
ax2d.legend()

#Grafico 2D - Angulos
ax_angle = fig.add_subplot(324)

ax_angle.set_xlim(0, 10)
ax_angle.set_ylim(-30, 30)

ax_angle.set_xlabel('Time (s)')
ax_angle.set_ylabel('Angulos (rad)')

phi_line, = ax_angle.plot([], [], lw=2, label='Phi',
                          color='blue')  # Linha Phi em azul
theta_line, = ax_angle.plot([], [], lw=2, label='Theta',
                            color='green')  # Linha Theta em verde
psi_line, = ax_angle.plot([], [], lw=2, label='Psi',
                          color='red')  # Linha Psi em vermelho
ax_angle.legend()

#Gráfico 2D - amplitudes dos Atuadores
ax_amp = fig.add_subplot(325)

ax_amp.set_xlim(0, 10)
ax_amp.set_ylim(33, 78)

ax_amp.set_xlabel('Time (s)')
ax_amp.set_ylabel('Amplitudes (m)')

amp_lines = [ax_amp.plot([], [], lw=2, label=f'L{i+1}')[0] for i in range(6)]
ax_amp.legend()

#dados para os gráficos 2D
times = np.linspace(0, 10, 500)
displacements = 1+ 0.3 * np.sin(3 * times)
#displacements = 1
phis = np.zeros_like(times)
thetas = np.zeros_like(times)
psis = np.zeros_like(times)



# Função de atualização para a animação
def update(num, B, L, lines, z_line, phi_line, theta_line, psi_line,
           amp_lines):
```

```
t = num / 50.0
zz =  0.3 * np.sin(3 * t)
trans = np.transpose(np.array([0, 0, zz]))  # X, Y, Z

phi =np.zeros_like(t)

theta =np.zeros_like(t)


rotation = np.transpose(np.array([phi, theta, 0]))  # X, Y, Z

R = np.matmul(np.matmul(rotX(rotation[0]), rotY(rotation[1])), rotZ(rotation[2]))
# Calculando os valores de cinemática inversa
l, lll, L, s = cinemática_inversa(trans, rotation, home_pos, R, P, B)

L = np.transpose(L)
# Atualiza os pontos da plataforma
lines['platform'].set_data(L[:, 0], L[:, 1])
lines['platform'].set_3d_properties(L[:, 2])

for i in range(6):
    lines['actuators'][i].set_data([B[0, i], L[i, 0]], [B[1, i], L[i, 1]])
    lines['actuators'][i].set_3d_properties([B[2, i], L[i, 2]])

for i in range(6):
    lines['base'][i].set_data([B[0, i], B[0, (i + 1) % 6]],
                              [B[1, i], B[1, (i + 1) % 6]])
    lines['base'][i].set_3d_properties([B[2, i], B[2, (i + 1) % 6]])

for i in range(6):
    lines['platform_edges'][i].set_data([L[i, 0], L[(i + 1) % 6, 0]],
                                        [L[i, 1], L[(i + 1) % 6, 1]])
    lines['platform_edges'][i].set_3d_properties(
        [L[i, 2], L[(i + 1) % 6, 2]])

# Atualiza os gráficos 2D
z_line.set_data(times[:num], displacements[:num])
#x_line.set_data(times[:num], phis[:num])
#y_line.set_data(times[:num], thetas[:num])

phi_line.set_data(times[:num], phis[:num])
theta_line.set_data(times[:num], thetas[:num])
psi_line.set_data(times[:num], psis[:num])

for i in range(6):
    amp_lines[i].set_data(times[:num], amplitudes[i, :num])
```

```
    return [
        lines['platform']
    ] + lines['actuators'] + lines['base'] + lines['platform_edges'] + [
        z_line, phi_line, theta_line, psi_line
    ] + amp_lines


# Configuração das linhas para o gráfico 3D
lines = {
    'platform': ax3d.plot([], [], [], 'o-')[0],
    'actuators': [ax3d.plot([], [], [], 'r-')[0] for _ in range(6)],
    'base': [ax3d.plot([], [], [], 'b-')[0] for _ in range(6)],
    'platform_edges': [ax3d.plot([], [], [], 'b-')[0] for _ in range(6)]
}

# Criação da animação
ani = animation.FuncAnimation(fig,
                              update,
                              frames=len(times),
                              fargs=(B, L, lines, z_line, phi_line, theta_line,
                                     psi_line, amp_lines),
                              interval=20,
                              blit=True)

plt.show()
```